SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 9, 365–378 (1997)

Practice

Software Maintenance Management Strategies: Observations from the Field

GEORGE E. STARK[1]* and PAUL W. OMAN[2]

[1]*The MITRE Corporation, 1150 Academy Park Loop #212, Colorado Springs CO 80910, U.S.A.*
[2]*Software Engineering Test Lab, University of Idaho, Moscow ID 83844–1010, U.S.A.*

SUMMARY

There is much literature describing software maintenance process models, but few comparative studies on the approaches used by managers in the field. This paper describes three software maintenance management strategies currently implemented by six organizations. The strategies are compared on the attributes of performance, economics, efficiency and end-product quality. The paper defines operational measurements for each attribute and describes the data collected over the past two years.

The three strategies investigated were labelled the fixed staff/variable schedule, variable staff/fixed schedule and the variable staff/variable schedule. Each strategy has attributes that make it appealing for implementation by a software maintenance project manager. The fixed staff/variable schedule strategy had the fastest priority change response time, the variable staff/fixed schedule strategy had the highest throughput and the best on-schedule performance, and the variable staff/variable schedule had the lowest cost and highest quality. Thus, the key task for a manager is defining the attribute that they would most like to optimize and choosing a strategy that supports that goal. © **1997 John Wiley & Sons, Ltd.**

J. Softw. Maint.: Res. Pract., **9**, 365–378 (1997)

No. of Figures: 8. No. of Tables: 2. No. of References: 18.

KEY WORDS: maintenance management; maintenance staffing; maintenance scheduling; maintenance strategies

1. INTRODUCTION

The importance of software maintenance in today's industry cannot be overestimated. It is widely recognized as the highest cost phase of the software life cycle with estimated costs of between 60% and 80% of the total software budget (Lientz and Swanson, 1978; Pigoski, 1997, pp. 29–31). Given this high cost, some organizations are looking at their maintenance process as an area for competitive advantage (Moad, 1990).

Several authors have noted that maintenance of software systems intended for a long operational life poses special management problems (Card, Cotnoir and Goorevich, 1987; Chapin, 1988; Hariza *et al*., 1992). The Software Engineering Institute (SEI) believes that

* Correspondence to: G. E. Stark, The MITRE Corporation, 1150 Academy Park Loop #212, Colorado Springs, CO 80910–3716, USA. E-mail: gstark@mitre.org

CCC 1040–550X/97/060365–14$17.50

Received 3 June 1997
Revised 29 August 1997

organizational processes are a major factor in the predictability of cost and quality of software (SEI, 1994). Furthermore, Alexander and Davis (1991) point out that today's project managers choose a software process model using *ad hoc*, unjustified and undocumented criteria. The selection of an inappropriate process model can result in

1. a release that does not satisfy user needs (reflected by the number of user-created problem reports and requirements volatility),
2. missed schedules with increased cost, and
3. more defects being released to the field.

According to IEEE Standard 1219, *IEEE Standard for Software Maintenance* (IEEE, 1993), software maintenance is the 'modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment'. At a macro level, two distinct activities occur during software maintenance: (a) requirements validation, and (b) maintenance release. The requirements validation process consists of reviewing individual change requests for their impact on the system as a whole. A change request can be either a defect correction or an enhancement to the system. A maintainer reviews the contents of the change request to ensure clarity, completeness, feasibility and consistency with outstanding change requests. Once the change request is determined to be a valid requirement, it is placed in a queue awaiting implementation during a maintenance release. Most organizations have a backlog of change requests available. While the requirement validation activity is central to the software maintenance process, it is not a focus of this paper.

Early published software maintenance release models viewed the implementation of changes from the point of view of the maintainer dealing with an individual change (Boehm, 1976; Martin and McClure, 1983; Sharpley, 1977; Parikh, 1980). Even more recent release descriptions (IEEE 1993; Basili *et al*., 1996) consider software maintenance as a one-change-at-a-time function. In our experience, however, a maintenance release is generally defined as a group of change requests based on the user priority, changes that affect the same subsystem, logical timing and other activities underway at the user location. Under this scenario, a management view is needed to understand the cost, quality and timeliness implications of different implementation strategies. We identified four release implementation strategies (hereafter termed management strategies) available to software maintenance managers:

- fixed staff/fixed schedule,
- fixed staff/variable schedule,
- variable staff/fixed schedule, and
- variable staff/variable schedule.

The overall objectives of this paper are: (1) to present software maintenance managers with the options available to them for implementing software releases, (2) to report our observations of these options in terms of four process criteria, and (3) to draw conclusions based on the first two objectives. Section 2 of this paper describes three of the four management strategies. The fixed staff/fixed schedule strategy was not implemented by

any of the six organizations that we observed, thus it is not described further. Section 3 identifies the key strategy criteria considered in this study (namely, performance, economics, efficiency and quality) and discusses the data collected. Section 4 presents the results and discusses their implications. Finally, Section 5 summarizes our conclusions and points out the limitations of our study.

2. MAINTENANCE MANAGEMENT OPTIONS

2.1. Three strategies observed

This section describes three management strategies for software maintenance organizations that reside several hundred miles from their users. According to Pigoski (1997, pp. 29–31), distance from the user has distinct advantages and disadvantages. The advantages include better documentation, formal release procedures and more clearly defined processes. The disadvantages are that significant training is required, user support can suffer and morale can decrease if the wrong people are assigned. We observed all of these effects on the maintenance programs we reviewed. Additionally, we observed that a strong measurement program supported the management throughout the maintenance process. Measurement helped managers to communicate more clearly with their customers on issues such as software cost estimates, requirements volatility, schedule constraints and product quality analysis.

Although there are four possible maintenance strategies (as listed earlier) only three of those strategies were actually used by the six software maintenance organizations we observed. The three strategies are labelled as (1) fixed staff/variable schedule, (2) variable staff/fixed schedule, and (3) variable schedule/variable staff. The fixed staff/fixed schedule strategy was not implemented by any of the six organizations, but the other three strategies provided a basis for comparison of the software maintenance life cycle. For each strategy it is assumed that the requirement validation activity is completed and the change request is available for incorporation in a version release. It is from this point in the maintenance life cycle that the three strategies are compared.

2.2. Fixed staff/variable schedule strategy

Figure 1 shows the implementation of the fixed staff/variable schedule strategy. In this implementation, there is a fixed pool of maintenance engineers available to the organization. A maintenance engineer takes a change request off the backlog queue and begins understanding, designing, coding and unit testing the change. The available staff works as many changes as possible through unit test until a ‘release freeze’ is declared by management. The release freeze is scheduled depending on the operational mission needs of the system, but is typically every six months for the systems we observed. Integration test time is scheduled on the operational platform for 30 days after the release freeze. All changes that the maintainers declare complete through unit test are then bundled into a release for integration test and formal turnover to the configuration management team. Note there is nothing inherent in the process that precludes code inspections or peer

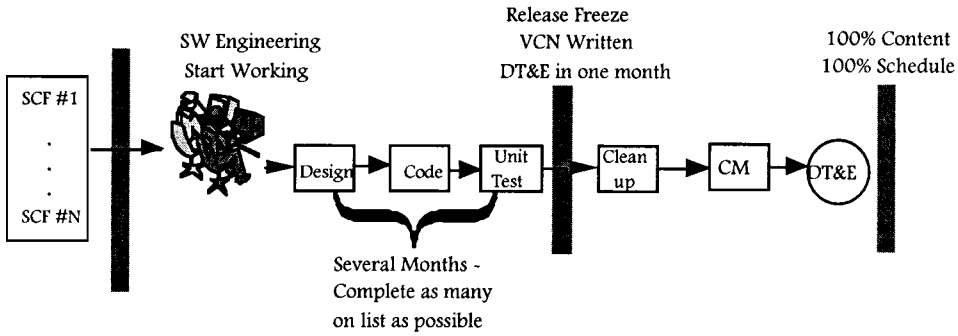

Figure 1. Diagram of the fixed staff/variable schedule software process

reviews, and some maintainers did perform them. They were not, however, formalized in the organizations that we observed.

Since the content is determined by the changes the maintainers have unit tested up to the release freeze date, and the release schedule is determined by the integration test schedule, this process should always contain 100% of the unit-tested content and always be delivered on time. The advantage of this process over the other two is its flexibility in choosing release content and the ease with which work is assigned.

2.3. Variable staff/fixed schedule strategy

Figure 2 depicts the flow of the variable staff/fixed schedule strategy. In this process, there may be a fixed pool of maintainers available, but they are allocated to individual software releases. The product release date is established between the customer and supplier based on a mission need date. Once the date is established and the 'high priority' changes are agreed for the mission, the system analysts prepare a preliminary version content notice (VCN) and a release plan. The preliminary VCN may contain additional changes to the software based on the change request backlog or the schedule. The maintainers begin working on the priority changes while the preliminary VCN and plan are being reviewed by management. If the maintainers or managers feel that the VCN is too ambitious or that some changes do not make sense within the context of the version,

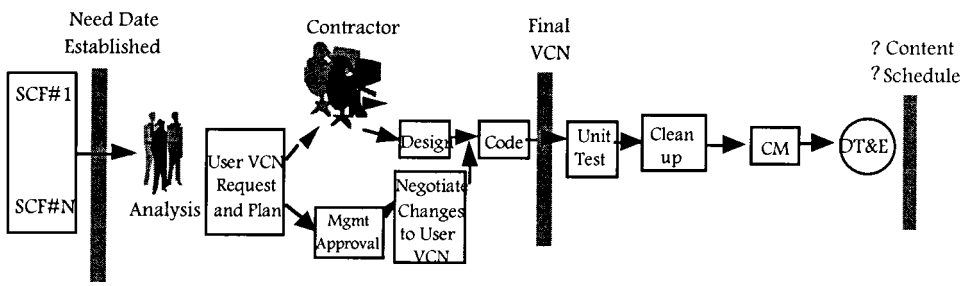

Figure 2. Diagram of the variable staff/fixed schedule process

the content or staffing may be renegotiated. A final VCN for the release is then issued prior to the completion of all unit testing. After unit testing the new version is turned over to configuration management for integration and operational testing prior to formal acceptance by the user.

Since the planned schedule is established between six and 18 months in advance, and some content is negotiable throughout the process, the actual content and delivery date vary with each release. The advantage this process has over the other two is its clearly defined need dates and priority changes. This forces the maintenance team to actively co-ordinate the release requirements with the user community and focuses the release on achieving the mission goals.

2.4. Variable staff/variable schedule strategy

The variable staff/variable schedule strategy is shown in Figure 3. In this strategy, there is not a fixed staff of maintainers, and the staff size and skill mix varies with each release. To determine the release content, system users submit a VCN request to the maintenance team. The maintenance team reviews the request and estimates the cost, schedule and risk associated with the requested change and subsequent release.

Based on this review, the maintenance team may either add content or negotiate less content with the users. After the content is determined, the maintenance team negotiates a release cost and schedule with a software maintenance contractor to design, code, test and integrate the release. Thus, the schedule is variable for each release in the sense that it is planned, but is not necessarily constrained by external influences. (This is different from the fixed staff/variable schedule strategy in which the release was not formally planned; and different from the variable staff/fixed schedule in that the schedule is not established prior to the content definition.) Milestone reviews are held during the implementation, and changes to the plan are made as necessary. The three technical reviews that occur in this process, but are not explicit in the other two processes, are an advantage because the reviews identify issues and help to ensure quality in the release.

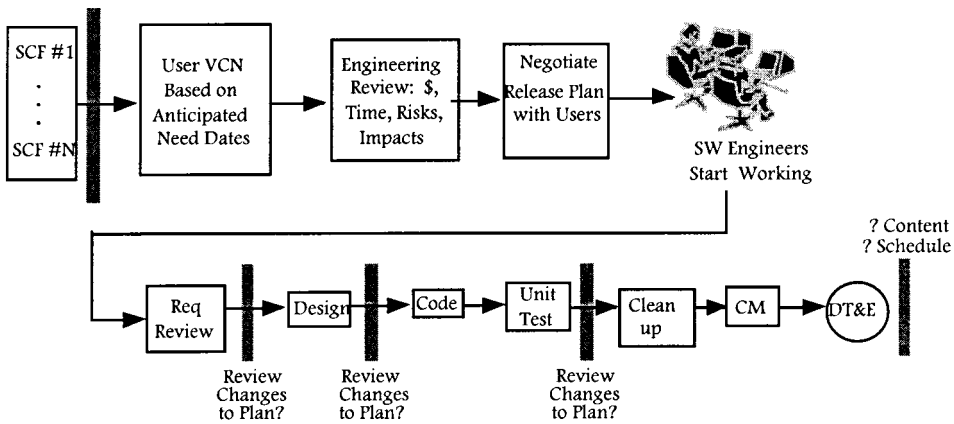

Figure 3. Diagram of the variable staff/variable schedule process

3. INDUSTRIAL PROFILES AND DATA CHARACTERISTICS

Data were collected between November 1994 and December 1996 on six different U.S. Department of Defense (DoD) maintenance organizations involved in maintaining real-time space-related applications and systems software (i.e., ground and space-based radar software, ground and on-board weather satellite software, and command and control software). All of the products are designed for use 24 hours a day, seven days a week and have been in the field for more than three years. The code under maintenance in each organization spanned several programming languages and ranged in size from 2 to 12 million lines of source code.

Some of the organizations are staffed completely by DoD personnel, others are a combination of contractor, civilian and DoD personnel. The major differences are the time constraints for the releases and the maintenance staff. Of the six organizations, three programs were using the fixed staff/variable schedule strategy; two were using the variable staff/fixed schedule strategy; and one was using the variable staff/variable schedule strategy.

Table 1 contains the salient characteristics of the organizations and products observed in this study. Actual measurements and further detail cannot be provided because the information is either proprietary or classified.

In this study there was no attempt to compensate for process differences, such as inspections or formal V&V (verification and validation) procedures. Some of the organizations participated in these activities, others did not. While we recognize the value of balancing or normalizing these differences, it was beyond the scope of this study to do so. Our activity was limited to observing existing processes.

Table 1. Characteristics of the organizations

Organization	Strategy	Software size (source lines of code)	Number of unique languages, primary language	Sites
A	Fixed staff/variable schedule	1 500 000	5, FORTRAN	2
B	Fixed staff/variable schedule	2 000 000	4, Ada	3
C	Fixed staff/variable schedule	1 800 000	3, Jovial	16
D	Variable staff/fixed schedule	12 000 000	28, Ada	3
E	Variable staff/fixed schedule	7 000 000	3, Ada	14
F	Variable staff/variable schedule	8 500 000	22, Jovial	10

4. STRATEGY SUCCESS CRITERIA

Basili and Rombach (1987) developed the goal/question/metric paradigm to help project managers tailor their software process to organizational goals and their operational environment. Alexander (1990) proposed a set of 20 criteria grouped into five categories (personnel, problem, product, resource and organization) to help a project manager select a process model. Whitten, Bentley and Barlow (1994) offer the PIECES criteria (Performance, Information, Economics, Control, Efficiency and Service) for improving systems and processes. Thus, the criteria that should be used in making strategy comparisons is open to debate.

The criteria presented here are by no means definitive and are not unique to software maintenance process models. However, the particular criteria were selected using literature research and the informed judgements of managers in the field with specific reference to the three maintenance strategies under observation. In some cases, the application may result in the elimination of inferior strategies. However, the relative weight to be placed on different criteria depends on the context in which the strategy is being implemented (as described above). Thus, all of the models may survive the comparison process, each with its own area of applicability.

The four criteria we chose for dependent measures were process performance, economics, process efficiency and product quality. The remainder of this section provides operational definitions for each of these criteria. We recognize that other, equally valid, definitions exist. The following definitions were chosen because they were the least intrusive under our observational constraints.

Process performance has two components: (1) throughput and (2) 'priority' cycle time. Throughput was measured as the average number of changes delivered to the user per year. The 'priority' cycle time was measured as the 75th percentile of the number of days required to deliver a priority change. The cycle time was calculated as

$$\textit{Cycle time} = \textit{Change delivery date} - \textit{Requirement validation date} \tag{1}$$

The 75th percentile was chosen because the cycle time distribution is skewed with the majority of the changes taking a short period of time, while a few take a long time to complete. Figure 4 is a cumulative distribution plot from one of the programs we observed.

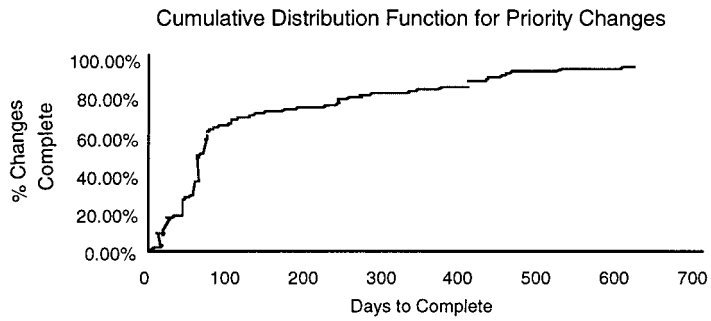

Figure 4. Cumulative distribution function of priority change request times observed for one of the observed programs

To read this plot, find 75% on the *y*-axis, (this is just to the right of the knee of the curve) read down to 135 days on the *x*-axis. This means that 75% of the time this strategy delivers high priority changes in less than 135 days from the time the requirement is validated. Note: the average for this distribution is 129 days with a standard deviation of 148 days.

For all three strategies, the priority of the change was determined by the system user as either emergency (highest priority), urgent (high priority) or routine (low priority). If the user determined the change was either emergency or urgent priority, its cycle time contributed to the percentile reported.

The economics of the process was calculated as the average cost per change delivered for each iteration of the maintenance process and subsequent release. Costs were tracked closely per release and included all expenditures including engineering, management, travel, configuration management, hardware maintenance on the software development system and financial overhead. The cost per change is calculated as

$$\textit{Cost per change} = \textit{Total Dollars spent} \div \textit{Total changes delivered} \tag{2}$$

For example, Figure 5 shows the total cost for each release completed during the 1996 fiscal year (FY96) and the first half of FY97 for one organization and the number of changes delivered in each release. From (2) the average cost per change is calculated by

$$\text{Cost (K\$)} = (\$13\,135 + \$12\,417 + \$12\,300) \div (197 + 183 + 108) = \$77{\cdot}56$$

The efficiency of the process is defined in terms of two attributes: (1) the per cent of schedules met, and (2) the per cent of releases that experienced requirements changes after the contents were defined. The per cent of schedules met is calculated as:

$$\textit{\% Schedules met} = \frac{\textit{(100)(Number of deliveries accepted on or before planned date)}}{\textit{(Total number of deliveries made)}} \tag{3}$$

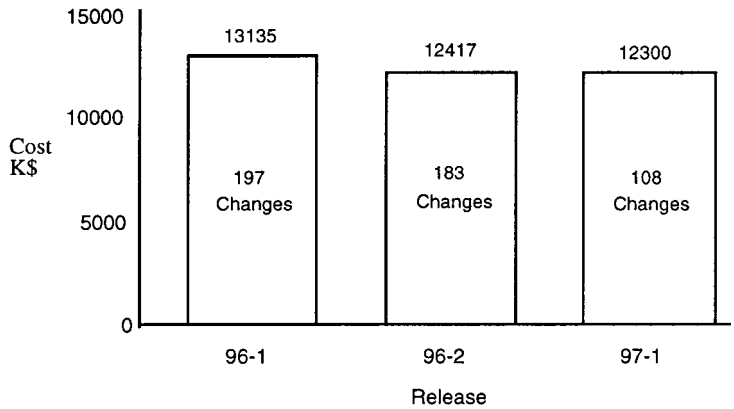

Figure 5. Cost per release for one program for a year and a half

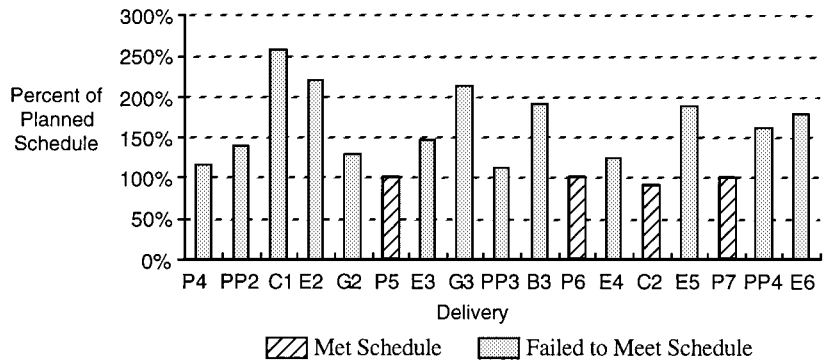

Figure 6. Schedule performance to plan for one program during FY95

Naturally, the optimum performance for each strategy is to meet 100% of the schedules. Figure 6 shows the raw data for one strategy. In this plot 100% means that the product was delivered on the planned date (e.g., P5), a value less than 100% means the product was delivered early (e.g., C2) and a value greater than 100% means the product was delivered late (e.g., C1). For this strategy, only 24% (four out of 17) of the planned delivery dates were met during the fiscal year plotted.

The second attribute of efficiency is the per cent of releases that experienced no content changes after the release plan was established. This is calculated as

$$\frac{\textit{Delivery efficiency} = (100)\ (\textit{Deliveries without content changes})}{(\textit{Total number of deliveries made})} \tag{4}$$

Naturally, the target for each process is 100%, but this is rare. In fact, Figure 7 shows that for this strategy eight of 13 deliveries experienced content change. Ambiguity in the requirements sometimes causes changes in scope. Customers changing their minds about priorities as well as schedule, and quality pressure often result in content additions or deletions from the plan.

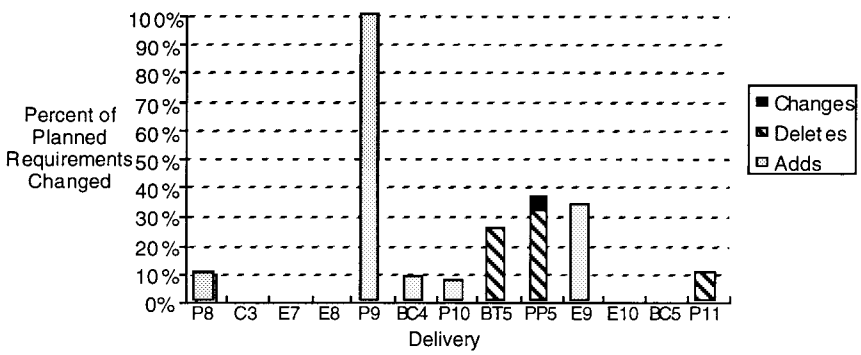

Figure 7. Example of data collected for requirements volatility by type for FY96

The quality of the product produced by each strategy was measured as the per cent of changes with customer found defects during operational acceptance testing. It is calculated as:

$$\text{Quality} = 1 - \left(\frac{(100)(\text{Changes with defects identified during customer testing})}{(\text{Total number of changes delivered})} \right) \quad (5)$$

Again, the goal here is 100%. The goal of each process is to deliver to the customer a software release that will execute the operational scenarios without failure. Customers often use different test scenarios from the developers and have different interpretations of failure, thus the metric goal was not completely met by any of the processes. Figure 8 shows the quality metric for four releases of one process. This is also a measure of the amount of rework required by the process since the defects found during customer testing are often written up as new change requests and must move through the process again.

5. RESULTS

Using the measurements defined in the previous section, the three maintenance management strategies were compared after two years of operation. Table 2 displays the results. The table contains a row for each strategy, a row that describes the goal for each measure and a column for each measurement collected. From the table, we observe the management trade-offs that should be considered in choosing a strategy. For example, a lower schedule achievement rate is associated with higher quality in the end product.

Table 2 shows the variable staff/fixed schedule had a higher throughput than the other two strategies. The strategy was also able to absorb a large percentage of requirements volatility and still meet 72% of their schedules. However, the variable staff/fixed schedule strategy had the highest cost per change and longest priority cycle time. This makes some sense since the schedule pressure and the requirements volatility require more staff and time to complete. Since this process is schedule driven, the urgent changes are made in the next available planning cycle, rather than incorporated into the current release. Because

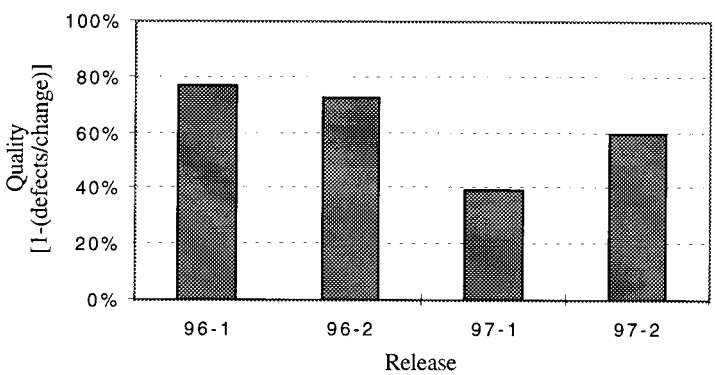

Figure 8. Example of the release quality metric for one project for two years

Table 2. Comparison of the management strategies

Criteria / Strategy	Process performance		Economics	Efficiency		Quality
	Average throughput in changes per year	Priority change response time in days	Dollars per change (in K$)	Per cent of schedules met	Per cent of releases with no content changes[1]	1 minus defects per change, as percentage
Goal/target	High	Low	Low	100%	100%	100%
Fixed staff/variable schedule (3 sites)	190	111	78	100%[2]	70% (10)	60%
Variable staff/fixed schedule (2 sites)	445	246	104	72%	22% (23)	71%
Variable staff/variable schedule (1 site)	136	135	41	43%	40% (30)	94%

Notes: 1—The total number of releases is shown in parentheses. 2—This is always 100% for this strategy.

releases are planned at least one year in advance, the cycle time is at least 365 days for urgent changes. The emergency changes (which are incorporated into the current release or become their own release) bring the cycle time down.

Although the variable staff/variable schedule strategy delivers the fewest changes per year, it appears the most cost-effective and it delivered the highest quality products over the time period. The higher quality is attributed to the impact of the multiple reviews and the addition of a requirements review missing from the other two processes.

As expected, the fixed staff/variable schedule process delivered the best 'on schedule' performance, while the variable staff/variable schedule process met only 43% of their plans. Interestingly, the fixed staff/variable schedule process showed 30% of the releases had a content change. This means that changes were added or deleted to the release during the 'freeze' period on three of the 10 releases observed.

6. CONCLUSIONS AND OBSERVATIONS

Three different software maintenance management strategies were described and compared on four attributes: performance, economics, efficiency and quality. Six measures were used to define the four attributes. All three processes have attributes that make them appealing to software maintenance managers:

- The fixed staff/variable schedule strategy allows managers to prioritize changes and allocate resources based on the available mix of personnel skills. It also supports the definition of delivery dates based on the organization's productivity.
- The variable staff/fixed schedule strategy handles requirements volatility and delivers a large number of changes per year.
- The variable staff/variable schedule strategy requires more formal planning and management oversight than the other two strategies, but experiences lower cost per change and highest quality (least number of defects per change).

Software maintenance organizations should choose the maintenance strategy which best suits their goals. We have observed these three strategies in production operations over a period of more than two years. In addition to the above conclusions we have made the following general observations which are related to maintenance staffing, quality and release schedules.

- Individual changes vary greatly in scope. For example, some changes are simple database updates while others require an 8 to 10 page specification and a thousand source lines of code to implement. This variability affects the economics of the strategy, the throughput and the quality. Because of the large number of total changes for each strategy, we assume that the statistical law of large numbers takes affect and that there was a similar distribution of change difficulty across the six programs.
- User expectations and participation in the release process differs. For one of the organizations we observed, the customer rarely participated in the version release process. Others had user teams supporting the planning and execution of the release. This participation affects the change prioritization, the release volatility and the

customer satisfaction with the release. We believe that customer participation should be encouraged in all strategies.

- The goals of the program management affect the outcome of the process. For example, if a program manager emphasizes flexibility, then the key measures for the strategy would be change response time or throughput. Alternatively, if a manager emphasizes economics, then cost per change becomes the key strategy measure.

Acknowledgements

We acknowledge Dave Card of Lockheed-Martin for his insightful discussions during the preparation of this paper. We also appreciate the *Journal's* anonymous reviewers for helpful comments.

References

Alexander, L. C. (1990) 'Selection criteria for alternate life cycle process models', Software Systems Engineering M. S. Thesis, George Mason University, Fairfax VA, 112 pp.

Alexander, L. C. and Davis, A. M. (1991) 'Criteria for selecting software process models', in *Proceedings of COMPSAC' 91*, Tokyo, Japan, IEEE Computer Society Press, Los Alamitos CA pp. 521–528.

Basili, V. R., Briand, L. C., Condon, S., Kim, Y., Melo, W. L. and Valett, J. D. (1996) 'Understanding and predicting the process of software maintenance releases', in *Proceedings of the 18th International Conference on Software Engineering*, Berlin, Gr., IEEE Computer Society Press, Los Alamitos CA, pp. 227–231.

Basili, V. R. and Rombach, H. D. (1987) 'Tailoring the software process to project goals and environments', in *Proceedings of the 9th International Conference on Software Engineering*, Monterey CA, IEEE Computer Society Press, Los Alamitos CA, pp. 345–357.

Boehm, B. (1976) 'Software engineering', *Transactions on Computers*, **C-25**(12), 1226–1242.

Card, D. N., Cotnoir, D. V. and Goorevich, C. E. (1987) 'Managing software maintenance cost and quality', in *Proceedings of the Conference on Software Maintenance—1987*, Austin TX, IEEE Computer Society Press, Los Alamitos CA, pp. 145–152.

Chapin, N. (1988) 'Software maintenance life cycle', in *Proceedings of the Conference on Software Maintenance—1988*, Phoenix AZ, IEEE Computer Society Press, Los Alamitos CA, pp. 6–13.

Hariza, M., Voidrot, J. F., Minor, E., Pofelski, L. and Blazy, S. (1992) 'Software maintenance: an analysis of industrial needs and constraints', in *Proceedings of the Conference on Software Maintenance*, Orlando FL, IEEE Computer Society Press, Los Alamitos CA, pp. 18–26.

IEEE (1993) *IEEE Standard for Software Maintenance*, IEEE Std 1219–1993, Institute of Electrical and Electronics Engineers, New York NY, 39 pp.

Lientz, B. P. and Swanson, E. B. (1978) 'Characteristics of application software maintenance', *Communications of the ACM*, **21**(6), 466–471.

Martin, J. and McClure, C. L. (1983) *Software Maintenance: The Problem and Its Solutions*, Prentice-Hall, Inc., Englewood Cliffs NJ, 472 pp.

Moad, J. (1990) 'Maintaining the competitive edge', *Datamation*, **36**(4), 61–62, 66.

Parikh, G. (1980) 'Some tips, techniques, and guidelines for program and system maintenance', in Parikh, G. (Ed), *Techniques of Program and System Maintenance*, Ethnotech, Inc., Lincoln NE, Winthrop Publishers, Cambridge MA, pp. 69–74.

Pigoski, T. M. (1997) *Practical Software Maintenance*, John Wiley & Sons, Inc., New York NY, 246 pp.

Sharpley, W. K. (1977) 'Software maintenance planning for embedded computer systems', in *Proceedings of COMPSAC'77*, IEEE Computer Society Press, Los Alamitos CA, pp. 520–526.

SEI (1994) *Software Process Maturity Questionnaire Capability Maturity Model, version 1.1*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, 32 pp.

Whitten, J. L., Bentley, L. D. and Barlow, V. M. (1994) *Systems Analysis and Design Methods*, 3rd edition, Irwin Publishing, Englewood Cliffs NJ, 480 pp.

Authors' biographies:

George E. Stark is a principal scientist with The MITRE Corporation in Colorado Springs where he supports the software efforts of the Missile Warning and Space Surveillance Sensors Program Management Office. His technical interests include software metrics and reliability for management decision-making. George has been involved in software reliability measurement for 13 years and was the vice-chairman of the AIAA blue-ribbon panel on software reliability. He has been the manager of software testing and reliability for a local loop fibre optic telephone system. He received the Johnson Space Center Quality Partnership award and the MITRE General Manager's award for contributions to software measurement. George received his bachelor's degree in statistics from Colorado State University and his master's degree in mathematics from the University of Houston.

Paul W. Oman is an Associate Professor of Computer Science at the University of Idaho, and an independent software consultant who specializes in software analysis. His consulting practice includes contract work for several international corporations and U.S. government agencies involved with the construction and maintenance of software for power production, medical instrumentation, environmental control systems, and personnel tracking and utilization. He occupies the Hewlett-Packard College of Engineering Research Chair at the University of Idaho, where he is Director of the Software Engineering Test Laboratory. He has a Ph.D. in computer science and is a member of the IEEE, the IEEE Computer Society and the ACM.

